Latency Measurement
for
KVM

Modification History

Revision	Date	Originator	Comments
0.1	01/26/11	Etienne Martineau	Initial draft

Table of Contents

1 Introduction

This document present information about Interruption latency as well as PCI pass through latency

1.1 Linux KVM; Background information

KVM is a type-2 hypervisor which is made two components. (A) KVM driver and (B) QEMU-KVM hypervisor.

1.1.1 KVM driver

The KVM driver manage un-privileged access to virtualization features (Intel vt-x) that can only be used directly by the privileged kernel.

1.1.2 QEMU-KVM hypervisor

The hypervisor user space process can request to create and configure a virtual machine (VM) by opening the KVM character device and issuing a series of IOCTLs. The execution context of the VCPU is implicitly defined by the user space hypervisor: It invoke the VCPU execution request from within one of it threads, and KVM driver preserves this scheduling context while running the guest code. Thus, if the Linux host scheduler decides to switch out a hypervisor thread it is the corresponding guest CPU which is effectively scheduled out. The hypervisor process can therefore apply standard means at the level of threads, processes, users to control groups to influence the scheduling of virtual machines.

The execution model is such that there is n VCPU threads together with one I/O processing loop, enabling more scalable SMP virtualization. As the QEMU core code is not yet prepared for multi-threaded execution, a global lock protects any access to the device emulation layer. Thus, only one VCPU or the I/O main loop may query or modify emulated devices of a virtual machine at the same time.

2 Test setup

In order to deliver an interruption to user-space, kernel must process the interruption vector first and then schedule the corresponding process.

The time it takes from when the interruption line is asserted to the point where the kernel process the vector is the *interruption latency*.

The time it takes from when the kernel schedule the corresponding process to the point where the process effectively runs is the *scheduling latency.*

Because in our environment we are relying on user-space driver, the interruption latency we are interested in is really: *interruption latency* + *scheduling latency* which in this documents is defined as *user interruption latency.*

2.1 Measuring the user interruption latency on the Host

High resolution timers has been used to measure the user interruption latency:

1) A user-space process sets up a timer to expire within n nanosecond and capture current time (T1)
2) The in-kernel 'timer wheels' configure the APIC timer to expire accordingly
3) The timer fire an interruption to the kernel.
4) Kernel eventually respond to the interruption (*interruption latency)*
5) Kernel schedule the user-space process and runs it (*scheduling latency*)
6) User-space capture current time (T2)

T2 – T1 gives the user interruption latency.

2.2 Measuring the user interruption latency on the Guest

Guest and host are based on native TSC clock source therefore results are coherent domain.

1) A guest user-space process sets up a timer to expire within n nanosecond and capture current time (T1)
2) The in-kernel 'timer wheels' configure the vAPIC timer to expire accordingly
3) On the host the timer fire and KVM 'inject' the interruption to the Guest kernel.
4) Guest kernel is scheduled in by the host kernel
5) Guest Kernel eventually respond to the interruption (*interruption latency)*
6) Guest Kernel schedule the user-space process and runs it (*scheduling latency*)

T2 – T1 gives the guest user interruption latency.

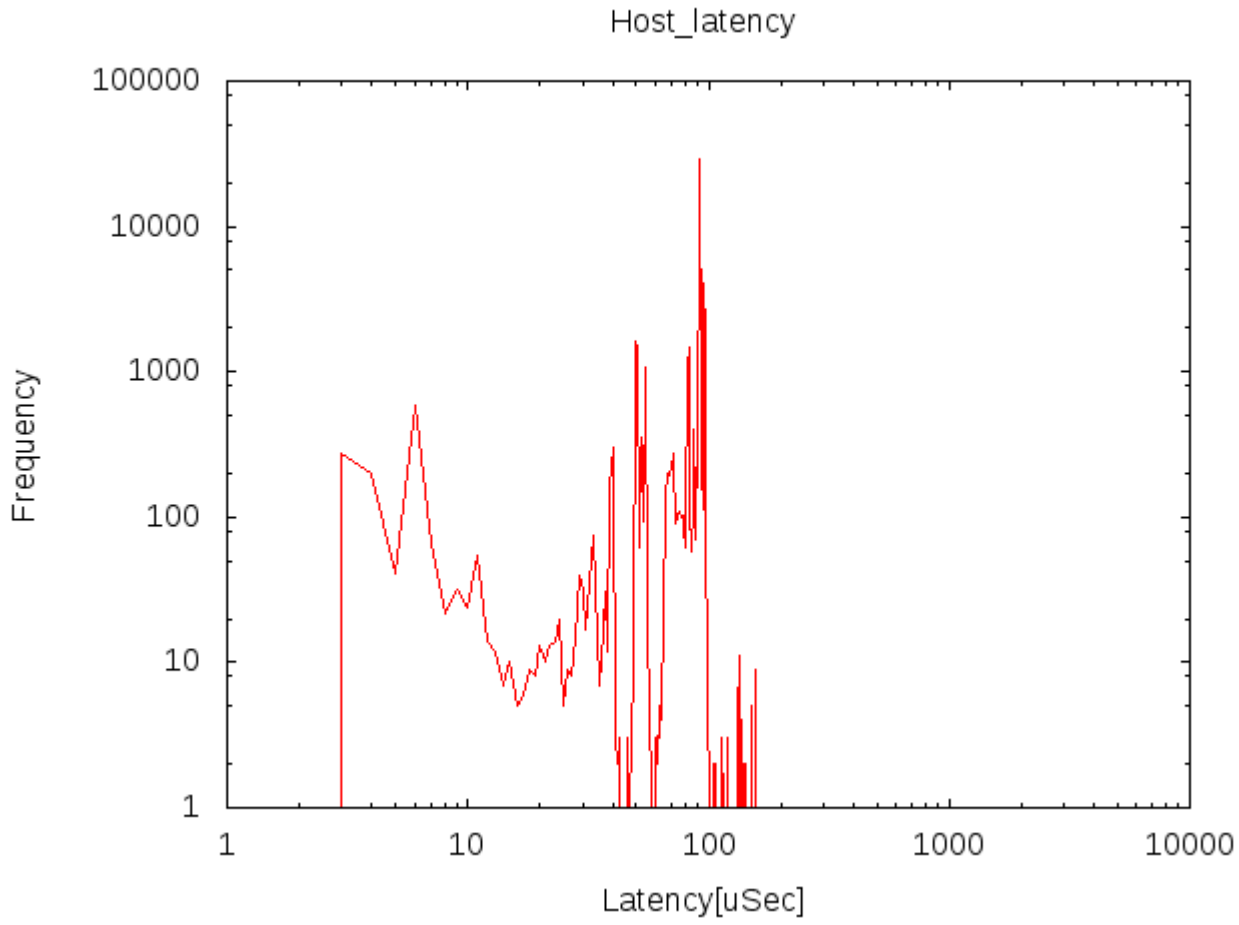
3 User interruption latency; Results

3.1 Host user interruption latency

Most of the distribution is around the 100 uSec range while there is still quite a few occurrence around the 50 uSec range. It's also interesting to observed that a latency less than 10 uSec is possible. The worst case scenario is ~150uSec.

Host:
Idle.

User-space process
Real-time task in RR 98.



3.2 Guest user interruption latency

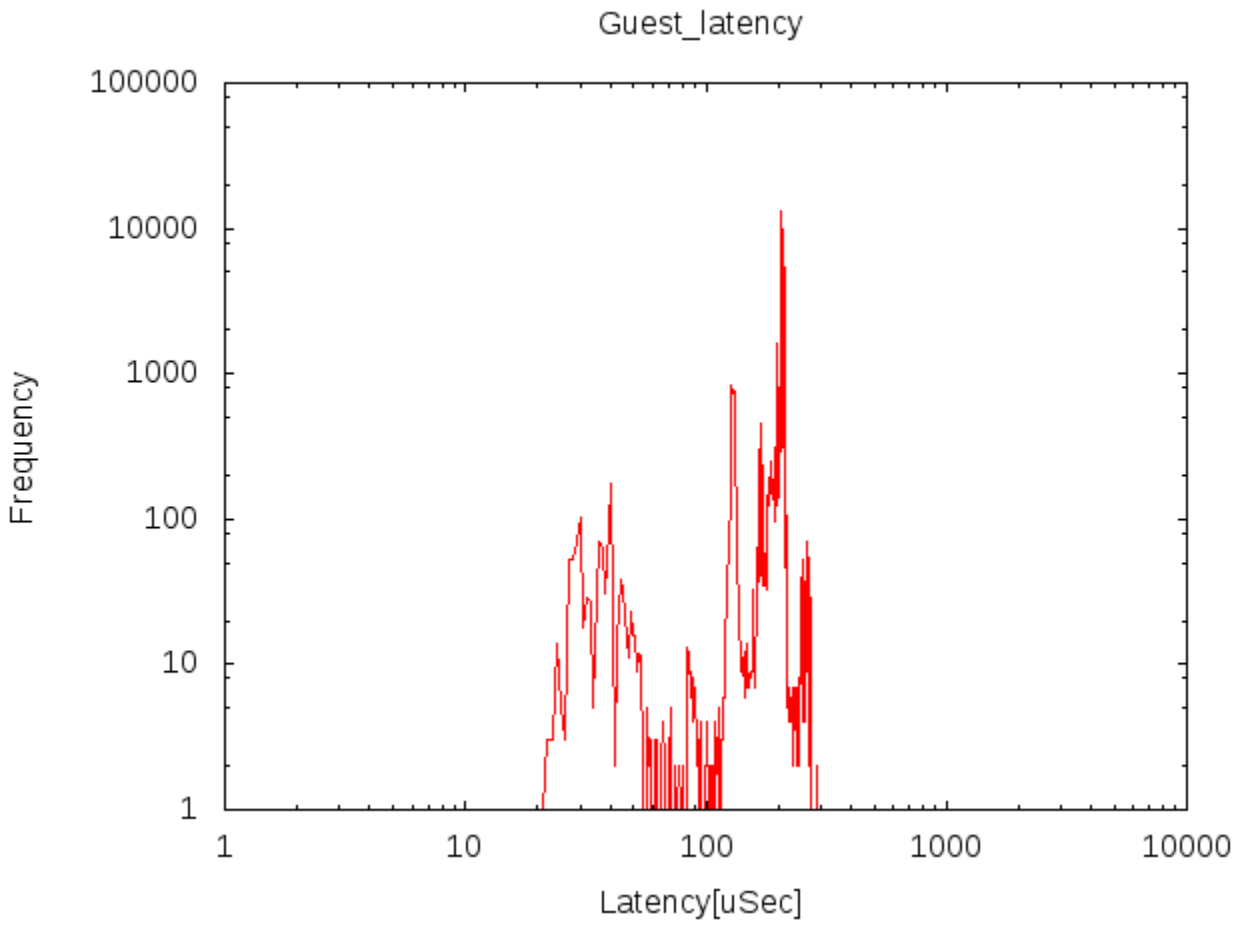
Most of the distribution is around the 200 uSec range while there is still quite a few occurrence around the 40 uSec range. There is no latency of less than 10 uSec. The worst case scenario is ~275uSec.

Host:

I/O thread: Sched FIFO 98; CPU = n
vCPU#1: Sched FIFO 97; CPU != n

Guest User-space process

Real-time task in RR 98.



4 Device pass-through

4.1 User interruption latency during device pass-through

For each test (1,2,3,4,5) the experience has been repeated 6 times

4.1.1 Results

Test #1 (Column A)

Lots of console IO operation
Host I/O: Sched other; **CPU = n**
Host vCPU#1: Sched other; **CPU != n**
Guest CPU#1 : Sched other
Max=1,303,922; Min=96,030

Test #2 (Column B)

Limited console IO operation
Host I/O: Sched other; **CPU = n**
Host vCPU#1: Sched other; **CPU != n**
Guest CPU#1 : Sched other
Max=8,013; Min=2,197

Test #3 (Column C)

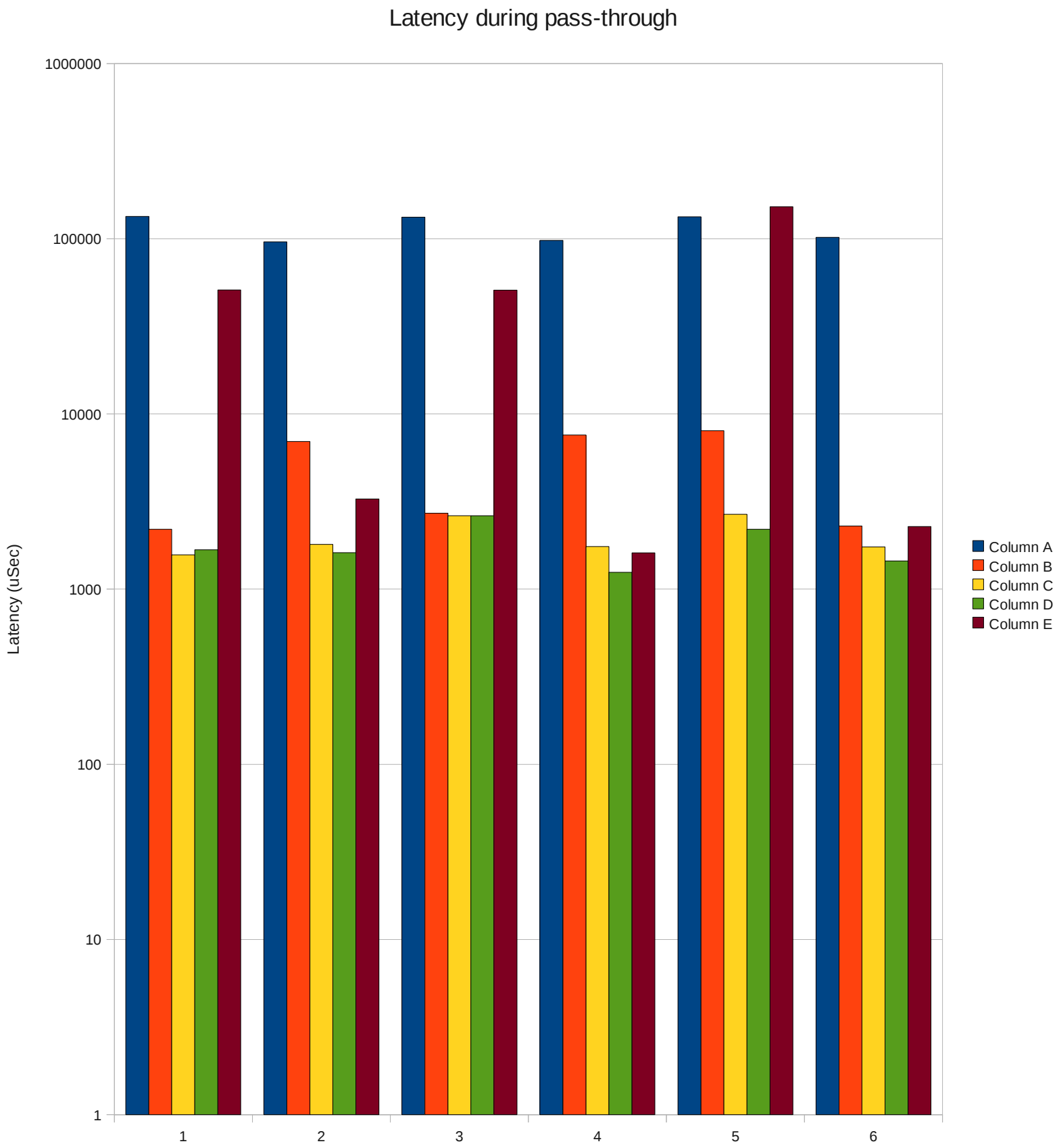
Limited console IO operation
Host I/O: Sched other; **CPU = n**
Host vCPU#1: Sched other; **CPU != n**
Guest CPU#1 : Sched RR 98
Max=2,672; Min=1,571

Test #4 (Column D)

Limited console IO operation
Host I/O: Sched RR 98; **CPU = n**
Host vCPU#1: Sched RR 97; **CPU != n**
Guest CPU#1 : Sched RR 98
Max=2,621; Min=1,247

Test #5 (Column E)

Limited console IO operation
Host I/O: Sched RR 98; **CPU = n**
Host vCPU#1: Sched RR 97; **CPU = n**
Guest CPU#1 : Sched RR 98
Max=152,335; Min=1,612


Latency during pass-through
Latency (uSec)
1000000
100000
10000
1000
100
10
1
1
2
3
4
5
6
Column A
Column B
Column C
Column D
Column E